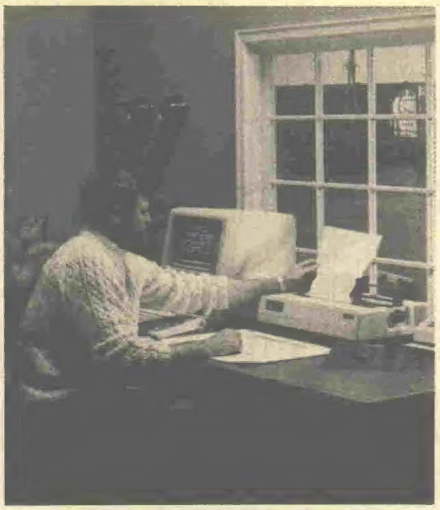
เมื่อจะเขียน คอมพายเลอร์

กิติพงษ์ ฤทธิบุตร

ที่แล้วๆมาเราก็ได้พูดกันถึงเรื่องโดยทั่วไปเกี่ยวกับคอมพายเลอร์ ทั้งในเรื่องของความหมาย โครงสร้างการทำงาน ภาษาที่ใช้ในการเขียน ขั้นตอนการทำงาน ความจำเป็นของการมีคอมพายเลอร์ ตลอดจนการนำคอมพายเลอร์บางภาษาที่มีใช้อยู่แล้วในระบบคอมพิวเตอร์ชนิดต่างๆมาดูกัน ว่าเขาได้ออกแบบไว้อย่างไร และคอมพายเลอร์เหล่านั้นทำงานกันอย่างไร คราวนี้เราจะมาลองดูกันว่าถ้าหากจะลงมือเขียนคอมพายเลอร์กันขึ้นมาจริงๆแล้ว สิ่งที่ควรจะรู้หรือกิจกรรมที่จำเป็นต้องทำ หรือสิ่งที่จะต้องคำนึงถึงนั้นประกอบด้วยอะไรบ้าง เพราะสิ่งต่างๆเหล่านี้จะช่วยทำให้เห็นโครงร่างหรือภาพรวมของกิจกรรมที่จะต้องทำในการออกแบบและเขียนคอมพายเลอร์ได้ชัดเจนยิ่งขึ้น

*บทความต่อจาก **คอมพายเลอร์ ล่าม คอมพิวเตอร์** (ไมโคร 24) และ **เปิดอกคอมพายเลอร์** (ไมโคร 28)

เนื้อหาหลักๆ ที่เราจะได้พูดถึงกันนั้น ก็จะเกี่ยวกับการทำความรู้จักกับความหมายของภาษาที่เกี่ยวข้องในการเขียนคอมพายเลอร์ (Language Definitions) เทคนิคการแปลแบบเรียกใช้ตัวเองตามลำดับขั้น (recursive descent) ที่เป็นที่นิยมใช้ในกระบวนการแปล การจัดการหรือการกำหนดรูปแบบโครงสร้างของแผนงานออกเป็นส่วนๆ (Intermodular Communication) เทคนิคการกระจายแบบแนวดิ่ง (Vertical Fragmentation) เทคนิคการกระจายแบบแนวราบ (Horizontal Fragmentation) และการประเมินผลในการสร้างคอมพายเลอร์ (Evaluation) ในการที่เราจะพูดกันถึงเรื่องต่างๆ เหล่านี้ก็จะเป็นไปในลักษณะของการแนะนำหรือพูดถึงในวงกว้างเพียงเพื่อให้ได้รู้จักหรือทำความเข้าใจไว้ในเรื่องเบื้องต้นเท่านั้น ซึ่งเราคงจะได้นำมาเป็นประเด็นร่วมกันศึกษาในโอกาสต่อไป



ภาษาคอมพายเลอร์ : เล่ากันใหม่

เราเคยได้พูดถึงภาษาต่างๆ ที่เกี่ยวข้องกับการเขียนคอมพายเลอร์กันมาบ้างแล้ว ซึ่งถ้าท่านผู้อ่านยังไม่ลืมกันก็คงจะนึกออกว่าประกอบด้วยภาษาต่างๆ หลายภาษา ได้แก่ภาษาที่ออกแบบเอาไว้ (source languages) ภาษาที่ต้องการหลังกระบวนการแปล (target languages) และภาษาที่ใช้ในการเขียนคอมพายเลอร์ (implementation languages) ว่าคืออะไร มีหน้าที่หรือความสำคัญในการเขียนคอมพายเลอร์อย่างไร ตลอดจนภาษาต่างๆ เหล่านี้มีความเกี่ยวข้องสัมพันธ์กันอย่างไร และแต่ละภาษามีภาษาอะไรบ้างที่จัดอยู่ในกลุ่มของตนเอง

การที่เรามีความเข้าใจเกี่ยวกับภาษาต่างๆ เหล่านี้ ก็จะทำให้สามารถแยกแยะได้ว่า ในแต่ละกลุ่มของภาษานั้นมีภาษาอะไรบ้าง และสามารถนำมาใช้ในขั้นตอนใด หรือมีวัตถุประสงค์ในการใช้อย่างไร ซึ่งจะทำให้ไม่เกิดความสับสนว่า ทำไมภาษาเดียวกันบางครั้งจึงเรียกเป็น source language ในขณะที่บางครั้งก็ถูกใช้เป็น implementation language หรือแม้แต่นำไปใช้เป็น target language และเหตุผลที่เกิดเหตุการณ์เช่นนั้นก็เนื่องจากว่า บางครั้งจะมีการนำภาษาระดับสูงบางภาษา เช่น อาจเป็น ALGOL 68 PASCAL หรือ C มาใช้เป็นภาษาในการเขียนคอมพายเลอร์ ในขณะที่แต่ละภาษาต่างก็เป็นภาษาที่จะต้องใช้คอมพายเลอร์ของตัวเองมาแปลให้เป็นภาษาเครื่องอีกต่อหนึ่ง หรือในทำนองเดียวกันอาจมีการนำเอาภาษา Assembly มาใช้เขียนคอมพายเลอร์ ในขณะที่การสร้างรหัสในขั้นสุดท้าย ก็ถูกออกแบบให้เป็นภาษา Assembly ด้วยก็ได้ เพราะฉะนั้นการที่จะมาทำความเข้าใจกับภาษาต่างๆ เหล่านี้ จึงนับได้ว่าเป็นสิ่งที่ไม่ควรจะมองข้ามไป

นอกเหนือจากภาษาต่างๆ ตามที่ได้กล่าวแล้ว อีกภาษาหนึ่งที่เราจะคุยกันเป็นการเพิ่มเติมในที่นี้ก็คือ ภาษาที่ใช้ในการอธิบายความหมายของภาษาอื่น ซึ่งเรียกว่า language defining languages หรือ definition languages โดยปกติแล้วภาษานี้จะใช้เพื่อการอธิบายหลักเกณฑ์ของภาษา(grammars) ที่ได้ออกแบบขึ้นมา อย่างไรก็ตามได้มีการนำไปใช้ในการออกแบบเพื่อให้ความหมายหรือคำนิยามของภาษาเป้าหมายอยู่เสมอ ซึ่งทำให้บางครั้งเกิดความสับสนขึ้นระหว่าง definition language และ implementation language เพราะมีการนำไปใช้งานในลักษณะเดียวกัน สำหรับความหมายที่ค่อนข้างชัดเจนของภาษานี้ก็คือ เป็นภาษาที่ใช้อธิบายหรือให้ความหมายของภาษาคอมพิวเตอร์ (programming language) ที่ออกแบบไว้นั้น ให้คอมพายเลอร์รู้ว่าจะต้องทำการแปลอะไรและอย่างไร แต่สิ่งที่สำคัญประการหนึ่งก็คือไม่ว่าจะเป็นภาษาที่มีรูปแบบอย่างไรก็ตามควรจะมีลักษณะที่สามารถสื่อหรือให้ผู้เขียนสามารถทำความเข้าใจได้ง่าย ในขณะที่สามารถใช้เป็นข้อมูล (input) ในการเขียนคอมพายเลอร์ได้ดีด้วย

เราลองมาดูตัวอย่างของ Formal Languages ที่ใช้สำหรับการอธิบาย Formal Grammars ซึ่งสามารถนำไปประยุกต์ใช้กับภาษาอะไรก็ได้ที่มีรูปแบบขึ้นอยู่กับหลักเกณฑ์ของภาษาตามที่ได้กำหนดขึ้นไว้ และมีวิธีการในการจัดรูปแบบโครงสร้างการใช้ได้หลายแบบแตกต่างกันออกไปตามกฎเกณฑ์ของภาษา เช่น การใช้ภาษานี้กับประโยคภาษาอังกฤษ โดยเรากำหนดให้มีโครงสร้างของภาษาซึ่งประกอบด้วย <sentence>, <subject>, <verb>, <object>, <noun> และ <article> ตามลำดับ และกำหนดต่อไปว่าให้มีคำ 6 คำที่ภาษานี้ยอมรับ คือ THE, BOY, GIRL SEES และ '.' (period) สำหรับกฎเกณฑ์ (rule) ของภาษานั้นกำหนดไว้ดังนี้

1. **<sentence> → <subject> <verb> <object>**
2. **<subject> → <article> <noun>**
3. **<object> → <article> <noun>**
4. **<verb> → SEES**
4. **<verb> → THE**
5. **<noun> → BOY**
7. **<noun> → GIRL**

ซึ่งกฎเกณฑ์เหล่านี้แสดงให้รู้ว่า <sentence> นั้นจะต้องประกอบด้วยการเริ่มต้นที่ <subject> ตามด้วย <verb> และ <object> โดยแสดงการสิ้นสุดของ <sentence> ด้วยเครื่องหมาย '.' ในขณะที่ <subject> จะประกอบด้วย <article> ตามด้วย <noun> สำหรับ <object> ก็จะเป็นลำดับของ <article> และ <noun> เช่นกัน รวมถึงการแสดงว่า <verb> นั้นได้แก่ SEES โดยมี THE เป็น <article> และ <BOY> กับ <GIRL> เป็น <noun>

จากโครงสร้างและกฎเกณฑ์ของภาษาตามที่ได้กำหนดไว้ข้างต้นนั้น เราสามารถที่จะกระจายออกตามลักษณะของ parse tree ได้ตามรูปที่ 1

ซึ่งจะเห็นได้ว่าโครงสร้างของภาษาที่เป็นที่ยอมรับว่าถูกต้องตามกฎเกณฑ์

COMPILE IN THE FAST LANE

นั้น จะต้องขึ้นต้นด้วย THE ซึ่งเป็น <article> ตามด้วย <noun> ซึ่งในที่นี้มีอยู่ 2 ตัวคือ BOY และ GIRL ตามด้วย SEES ที่ทำหน้าที่แทนตำแหน่งของ <verb> ต่อด้วย THE กับ <noun> และจบลงด้วยเครื่องหมาย '.'

ในขั้นตอนต่อไปที่จะพิจารณาว่ารูปแบบของประโยคที่ภาษาที่ยอมรับนั้นมีอะไรบ้าง ก็สามารถทำได้โดยการแทนที่ <noun> ตามกฎเกณฑ์ที่ได้กำหนดไว้ นั่นคือรูปแบบที่ถูกต้องจะมีเพียง 4 ประโยคเท่านั้นได้แก่

1. THE BOY SEES THE GIRL.
2. THE BOY SEES THE BOY.
3. THE GIRL SEES THE BOY.
4. THE GIRL SEES THE GIRL.

ซึ่งหมายความว่าถ้ารูปแบบของประโยคที่ได้ไม่เป็นไปตามสี่ประโยคข้างต้นนี้แล้ว ก็จะถือได้ว่าผิดกฎของภาษานี้

ภาษาในลักษณะนี้ที่เป็นที่รู้จักและนำมาใช้กันอยู่เสมอๆ ในการเขียนคอมพายเลอร์นั้นก็ได้แก่ 0 language หรือ L(G) language ซึ่งเป็นภาษาที่สร้างขึ้นจาก (CSL) ซึ่งใช้กับโครงสร้างของภาษาแบบ context-sensitive grammar (CSG) นอกจากนี้ก็ยังมีภาษาในแบบ context-free language (CFL) ที่สร้างขึ้นโดยหลักเกณฑ์แบบ context-free grammar (CFG) และก็ยังมีภาษาในรูปแบบที่เรียกว่า regular language (RL) ซึ่งเป็นภาษาที่ใช้กับโครงสร้างของภาษาแบบ regular grammar (RG) เป็นต้น สำหรับหลักเกณฑ์ของภาษาหรือวิธีการนำไปใช้จะได้พูดถึงต่อไปเมื่อมีการอ้างถึง อย่างไรก็ตามผู้ที่จะเขียนคอมพายเลอร์ก็ควรจะศึกษาและทำความเข้าใจกับสิ่งสำคัญๆ เหล่านี้เพื่อใช้เป็นพื้นฐานในการออกแบบโครงสร้างของภาษาต่อไป

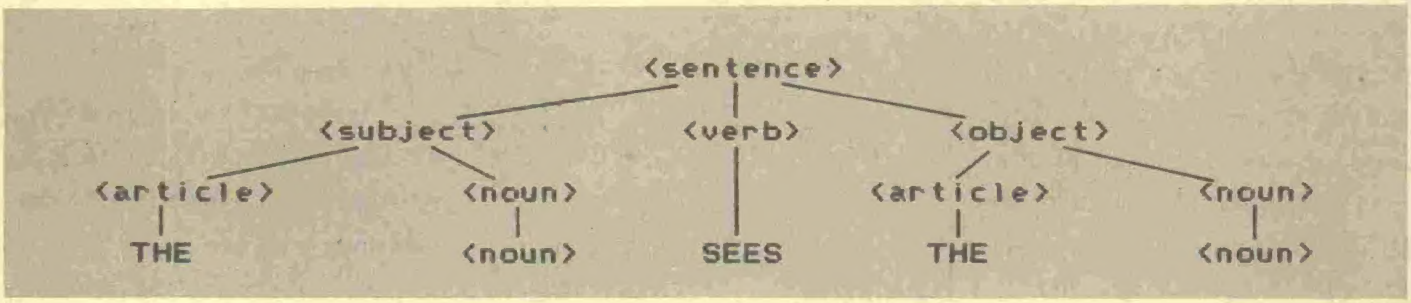
รีเคอร์ซีฟอีกที

เรื่องเทคนิคการแปลที่มีการเรียกใช้ตัวเองแบบรีเคอร์ซีฟนั้น เราเคยได้พูดถึงกันมาแล้วในครั้งก่อนๆ เพราะฉะนั้นคราวนี้ก็จะได้กล่าวถึงเพิ่มเติมในส่วนที่ยังไม่ได้อ้างถึง

เทคนิคแบบนี้อาจจะพูดได้ว่าเป็นเทคนิคที่ไม่ใช่สิ่งแปลกใหม่สำหรับนักเขียนโปรแกรม เพราะเป็นสิ่งที่รู้จักกันอย่างแพร่หลาย และมีการนำมาใช้ในการทำงานจริงๆ กันอยู่เสมอ อย่างที่เราเคยคุยกันมาแล้วว่าได้มีการนำเทคนิคนี้มาใช้ในการเขียน C compiler เป็นต้น เพราะเป็นวิธีการที่ไม่ยุ่งยากนัก และสามารถนำมาใช้เพื่อเขียน/สร้างตัวแปลภาษาที่ดีได้

การที่มีผู้นำวิธีการแปลแบบรีเคอร์ซีฟมาใช้กันอย่างแพร่หลายนั้น ก็เนื่องจากเหตุผลหลายอย่างที่เป็นคุณสมบัติเฉพาะตัวของเทคนิคแบบนี้ ซึ่งพอจะสรุปออกเป็นประเด็นๆ ได้ดังนี้

- เทคนิคแบบการเรียกใช้ตัวเองนั้น เป็นวิธีที่สามารถทำได้ไม่ยากนัก ซึ่งผู้เขียนโปรแกรมที่มีประสบการณ์ในการเขียนโปรแกรม สามารถที่จะทำความเข้าใจได้ง่าย และนำมาใช้งานได้เร็วกว่าวิธีอื่นๆ อีกหลายวิธี



รูปที่ 1 วิธีการตรวจสอบ grammar ด้วย parse tree

● คุณสมบัติที่ดีอีกอย่างหนึ่งของเทคนิคนี้ก็คือ การที่ผู้เขียนโปรแกรมสามารถจะทำการเพิ่มเติมส่วนของการสร้างรหัสระหว่างประโยคคำสั่งในส่วนที่ทำหน้าที่ตรวจสอบความถูกต้องของรูปแบบภาษาตามที่กำหนดไว้(recognizer) ได้ ซึ่งหมายความว่าในขณะที่อยู่ในระหว่างขั้นตอนการตรวจสอบโครงสร้างของข้อมูลจากโปรแกรมต้นฉบับ (source program) นั้น ผู้เขียนโปรแกรมมีโอกาสที่จะทำการแทรกส่วนที่ทำหน้าที่ในการแปลเข้าไปด้วยได้อย่างสะดวก เพื่อทำการแปลคำสั่งจากโปรแกรมต้นฉบับให้เป็นรหัสที่ต้องการ

● ข้อดีอีกประการหนึ่งก็คือ การที่ตัวแปรที่ใช้เฉพาะส่วน (local variables) ของกระบวนการทำงานที่เรียกใช้ตัวเองนั้น จะถูกสร้างขึ้นในสแต็ค ระหว่างการประมวลผล (run-time stack) หมายความว่าเมื่อมีการอ้างถึงหรือเรียกใช้ตัวแปรใดๆ ในขณะประมวลผล ระบบก็จะทำการจัดสรรเนื้อที่ในหน่วยความจำหลักให้กับตัวแปรนั้น ตามที่ได้กำหนดรูปแบบไว้แต่แรก ซึ่งทำให้การเรียกใช้ตัวแปรเหล่านั้นทำได้อย่างสะดวก โดยที่ผู้เขียนโปรแกรมไม่จำเป็นต้องเขียนโครงสร้างข้อมูลแบบสแต็ค (stack data structures) ไว้ในขั้นตอนการทำงานเลย

จะเห็นได้ว่าวิธีการเรียกใช้ตัวเองแบบนี้ เป็นวิธีการที่ให้ความสะดวกในการเขียนคอมพายเลอร์ได้อย่างมากวิธีหนึ่ง และส่วนที่น่าจะเป็นสิ่งได้เปรียบมากที่สุดของวิธีนี้ก็คือ ความง่ายและขีดความสามารถของเทคนิคที่จะทำให้การแปลภาษาต้นฉบับที่มีความซับซ้อนเป็นไปได้อย่างราบรื่น และมีความสะดวกในการแปลรหัสมากขึ้น

อย่างไรก็ตามการใช้เทคนิคแบบรีเคอร์ซีฟในการเขียนคอมพายเลอร์นั้นก็มีสิ่งจำกัดบางประการ ซึ่งนับว่าเป็นข้อเสียที่สำคัญของเทคนิคแบบนี้ โดยมีสาเหตุเนื่องมาจากการที่วิธีการนี้มีส่วนของการสร้างรหัส (code generator) และการตรวจสอบความถูกต้องของโครงสร้างภาษา (parser) อยู่ในขั้นตอนการทำงานชุดเดียวกัน ซึ่งทำให้ไม่สามารถที่จะเข้าไปจัดการในลักษณะของการแยกแต่ละส่วนเพื่อวัตถุประสงค์ต่างๆ ได้ เช่นในเรื่องเกี่ยวกับเอกสาร (documentation) การบำรุงรักษา (maintenance) การจัดการหน่วยความจำหลัก (memory management) การทดสอบ (testing) และอื่นๆ อีกหลายประการ และจากสาเหตุนี้ก็จะส่งผลให้การสร้างโปรแกรมภาษาเครื่อง (target code) ไม่มีความเป็นอิสระต่อเครื่อง

การเขียนคอมพายเลอร์นั้นมีการนำเทคนิคแบบให้มีการเรียกใช้ตัวเองมาใช้กันอย่างแพร่หลาย และอาจกล่าวได้ว่าวิธีการนี้เป็นเครื่องมือที่มาตรฐานอย่างหนึ่งในการเขียนคอมพายเลอร์ และต่อไปเราจะได้ลองมาดูวิธีการใช้ในงานบางลักษณะ เช่น ในคำสั่งแบบมีเงื่อนไข (conditional statements) และการตรวจสอบความถูกต้องตามกฎเกณฑ์ของภาษา (syntax analysis) เพื่อเป็นแนวทางในการนำไปใช้งานจริงในโอกาสข้างหน้า

คำสั่งแบบมีเงื่อนไข

ที่นี้เราลองมาดูกันว่า การนำเทคนิคแบบรีเคอร์ซีฟมาใช้กับคำสั่งประเภทที่มีเงื่อนไขนั้นทำได้อย่างไร แต่ก่อนที่จะไปถึงวิธีการนี้ เราลองมาดูหลักเกณฑ์ในการทำงานในลักษณะที่มีเงื่อนไขกันก่อน ก็มาสมมุติกันว่ามีประโยคคำสั่งดังนี้

IF B, S1, S2

ซึ่งในที่นี้หมายความว่า ให้ทำการกระโดดไปทำงานที่คำสั่ง S1 ถ้า B เป็นจริง และ ให้ไปทำที่คำสั่ง S2 ถ้า B เป็นเท็จ ในกรณีนี้การแปลรหัสก็จะต้องแปลความหมายให้อยู่ในรูปแบบตามรูปที่ 2

และเมื่อคอมพายเลอร์อ่านประโยคคำสั่งที่ขึ้นต้นด้วย IF ก็จะทำการเรียกใช้โปรแกรมย่อย (subroutine) ซึ่งในส่วนของการทำงานของโปรแกรมนี้จะทำการอ่านคำสั่งและแปลความเพื่อสร้างรหัสให้มีการทำงานตามคำสั่งต่อไป และวิธีการเรียกใช้ตัวเองก็จะใช้วิธีการแบบนี้ แต่จะเป็นการเรียกใช้ตัวเองแทนการเรียกใช้โปรแกรมย่อยอื่นต่อไป เราลองมาดูกัน

```
        Jump to L1 if not - B
        S1
        Jump to L2
L1 :    S2
L2 :    .....
```

รูปที่ 2
แสดงการสร้างรหัสจากคำสั่งที่มีเงื่อนไข

COMPILE IN THE FAST LANE

ว่าการเรียกใช้ตัวเองด้วยวิธีการข้างต้นนั้นเป็นอย่างไร เราก็มาสมมุติกันต่อไปว่ามีคำสั่ง IF ที่ต่อเนื่องกันดังนี้

IF B1, IF B2, S1, S2, S3

ซึ่งความหมายของประโยคคำสั่งข้างบนนั้น จะมีความหมายว่า ถ้า B1 เป็นจริง ให้ทำ IF B2 ถ้าเป็นเท็จให้ทำ S3 โดยจะทำ S1 เมื่อ B2 เป็นจริง และทำ S2 ถ้า B2 เป็นเท็จ เราจะเขียนคอมพายเลอร์เพื่อแปลความหมายของประโยคคำสั่งเหล่านี้ด้วยการเขียนโปรแกรมย่อยโดยให้ชื่อว่า COMPILE และให้มีขั้นตอนการทำงานเป็นลำดับตามที่จะได้แสดงตามรูปที่ 3

จากตัวอย่างจะเห็นได้ว่าการทำงานของโปรแกรม COMPILE นั้น มีการเรียกใช้ตัวเองในกรณีที่ได้ตรวจพบว่ามีคำสั่ง IF และในกรณีของตัวอย่างนี้ เมื่อโปรแกรม COMPILE ตรวจสอบคำสั่ง IF และพบว่ามี IF ตัวที่สองก็จะทำการแปลคำสั่งและสร้างรหัสตามเงื่อนไข

```
COMPILE :   if stalement begins 'IF', go to LL
            compile   stalement  (terminated  by  comma or
                      newline)
            exit
LL :        read condition (up to comma)
            compile  code  to  evaluate  inverse condition
            issue labels L1, L2
            compile 'Jump L1 if true'
            call COMPILE
            compile ' Jump L2
                     L1:'
            call COMPILE
            compile 'L2:'
            Exit
```

รูปที่ 3 แสดงขั้นตอนการทำงานแบบรีเคอร์ซีฟ

ของ IF ตัวที่สองนี้ให้เรียบร้อยก่อนแล้วจึงกลับไปทำงานในส่วนของโปรแกรม COMPILE ชุดก่อนๆ ต่อไปจนเสร็จ ซึ่งรหัสคำสั่งที่ได้จากการแปลนั้นจะมีการกำหนดตำแหน่งการกระโดดให้กับคำสั่ง IF ด้วย และขั้นตอนการทำงานของรหัสก็จะเป็นไปตามที่ได้แสดงไว้ในรูปที่ 4

จะเห็นได้ว่าในกระบวนการแปลความของโปรแกรม COMPILE ซึ่งจะถูกเรียกใช้ทุกครั้งที่เจอคำสั่งที่ขึ้นต้นด้วย IF นั้น จะทำหน้าที่ตรวจสอบเงื่อนไขของ IF และทำการสร้างรหัสที่เป็น target code ต่อไป ตามตัวอย่างข้างต้น โปรแกรม COMPILE จะทำการสร้างรหัสสำหรับคำสั่งประเภทที่มีเงื่อนไขซ้อนกัน ดังนั้นภายในโปรแกรมจึงต้องเรียกใช้ตัวเองอีกสองครั้ง เพื่อจะทำการสร้างรหัสตามคำสั่งได้ครบถ้วน

สำหรับเทคนิคการเขียนโปรแกรมรีเคอร์ซีพนี้ มีการนำไปใช้กันอย่างกว้างขวาง และอาจจะพูดได้ว่าเป็นเครื่องมือที่ค่อนข้างจะมาตรฐานสำหรับการเขียนคอมพายเลอร์ ซึ่งก็หมายความว่า ถ้าเราจะเขียนคอมพายเลอร์สักตัวหนึ่ง ก็คงจะต้องมาทำความเข้าใจกับเทคนิคนี้กันก่อน

เกี่ยวกับโมดูล

เทคนิคอีกประการหนึ่งที่มีการนำมาใช้ในการเขียนโปรแกรมคอมพายเลอร์ก็คือการแบ่งโปรแกรมออกเป็นส่วนย่อยๆ (modular programming techniques) ซึ่งหมายความว่างานบางส่วนของโปรแกรมนั้น สามารถแยกเขียนต่างหากจากโปรแกรมหลัก หรือทำการเปลี่ยนแปลงแก้ไข ทดสอบและอื่นๆ ได้อย่างอิสระ โดยจะนำแต่ละส่วนมาเชื่อมโยงกันในขั้นสุดท้ายหรือในการประมวลผลจริง เพื่อทำงานอย่างใดอย่างหนึ่งตามความต้องการของผู้ออกแบบโปรแกรมนั้น ซึ่งวิธีการนี้นับเป็นวิธีที่มีประโยชน์มาก การเขียนโปรแกรมโดยเฉพาะอย่างยิ่งโปรแกรมที่มีขนาดใหญ่ และมีวัตถุประสงค์ที่จะให้ทำงานได้หลายๆ อย่าง การแยกโปรแกรมออกเป็นส่วนๆ (module) จะช่วยให้สามารถกระจายงานเขียนโปรแกรมให้กับคนหลายๆ คนช่วยกันทำ ซึ่งจะมีผลให้งานเสร็จเร็วขึ้น และถ้าการมอบหมายงานนั้นเป็นไปอย่างเหมาะสมคือ ให้งานกับบุคคลที่มีความสามารถเฉพาะด้าน (มีความชำนาญในเรื่องที่มอบหมายให้) ก็จะทำให้ผลงานที่ได้มีคุณภาพดียิ่งขึ้น นอกจากนี้เทคนิคแบบนี้ยังมีประโยชน์ต่อการดูแลรักษา ตลอดจนสามารถทำการแก้ไขพัฒนาโปรแกรมได้อย่างสะดวก รวมถึงการจัดทำเอกสารประกอบเพื่ออธิบายโปรแกรมแต่ละส่วนเหล่านั้น โดยสามารถแสดงรายละเอียดต่างๆ เกี่ยวกับโปรแกรมได้อย่างเด่นชัด

งานด้านการเขียนคอมพายเลอร์เป็นงานที่ต้องการความประณีตและเป็นงานใหญ่ที่ต้องการบุคลากรที่มีความรู้ความสามารถในการเขียนโปรแกรมอย่างมากงานหนึ่ง(ทั้งนี้จะมากน้อยเพียงใดก็ขึ้นอยู่กับโครงสร้างของภาษาต้นฉบับที่ออกแบบไว้ และขีดความสามารถของคอมพายเลอร์ที่ต้องการ) ดังนั้นจึงได้มีการนำเทคนิคนี้มาใช้กับงานเขียนโปรแกรมคอมพายเลอร์ด้วยเช่นกัน และได้นำมาใช้กับการดำเนินงานในทุกขั้นตอน ไม่ว่าจะเป็นในเรื่องที่เกี่ยวกับรูปแบบของภาษาต้นฉบับ ภาษาเครื่องที่ต้องการและกระบวนการแปลที่แบ่งออกเป็นส่วนย่อยๆ เป็นต้น ซึ่งจะเห็นได้ว่าอย่างน้อยงานที่ควรจะนำเทคนิคแบบแยกงานออกเป็นส่วน ๆ มาใช้ก็ควรจะประกอบด้วยงานสี่ประเภทได้แก่ งานด้านเอกสารประกอบงานเขียนคอมพายเลอร์ งานเขียนที่มอบหมายให้กับผู้เขียนโปรแกรมแต่ละคน งานเกี่ยวกับการให้รายละเอียดของภาษาที่ได้ออกแบบไว้ และงานในส่วนของรูปแบบภาษาเครื่องที่ต้องการสำหรับคอมพายเลอร์ จะเห็นได้ว่าการที่ทำการออกแบบโดยแยกงานออกเป็นส่วนๆ นั้น จะช่วยให้กระบวนการต่างๆ ที่เกี่ยวข้องกับการเขียนโปรแกรมคอมพายเลอร์ดูง่ายขึ้น อย่างไรก็ตามการแบ่งงานออกเป็นส่วนๆ นี้ แต่ละงานแต่ละคนย่อมสามารถที่จะออกแบบอย่างไรก็ได้ ไม่มีกฎเกณฑ์ตายตัวว่าจะต้องทำอย่างไรมากน้อยแค่ไหน แต่ขึ้นอยู่กับความเหมาะสมของงานและการตัดสินใจของผู้ออกแบบเอง การที่ได้แบ่งส่วนไว้สี่ประเภทนั้นเป็นเพียงการแบ่งอย่างกว้างๆ เพื่อเป็นแนวทางเท่านั้น ที่นี้เราลองมาดูกันว่าเทคนิคแบบแยกงานออกเป็นส่วนๆ นั้นจะนำมาใช้กับงานชนิดต่างๆ ตามที่ได้

```
        Jump L1 if not - B1
        Jump L3 if not - B2
        S1
        Jump L4
L3:     S2
L4:     Jump L2
L1:     S3
L2:     ....
```

รูปที่ 4
แสดงขั้นตอนการสร้างรหัสจากคำสั่งแบบเงื่อนไข

COMPILE INITHE FASTILANE

แบ่งประเภทไว้อย่างไร

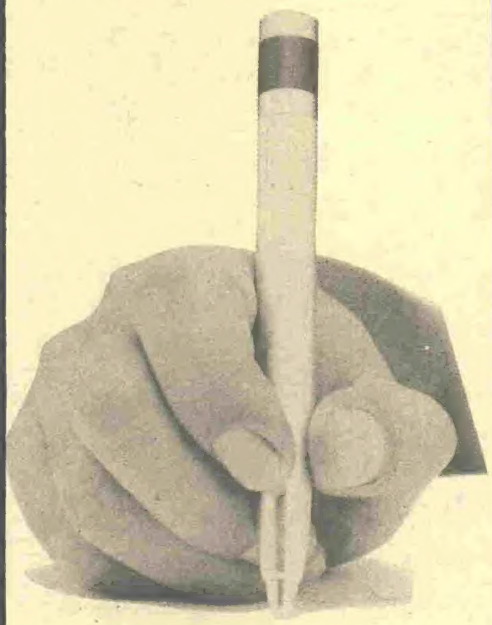
การนำเทคนิคการแยกเป็นส่วนๆ มาใช้กับงานด้านเอกสาร (modular documentation) นั้น ก็เป็นการใช้ในลักษณะของการอธิบายหรือให้รายละเอียดเกี่ยวกับงานเขียนคอมพายเลอร์ ไม่ว่าจะเป็นงานที่เกี่ยวกับการวางแผน การออกแบบหรือการบันทึกขั้นตอนการทำงาน ซึ่งเป็นงานที่อาจต้องใช้เวลาสำหรับงานด้านนี้มากพอควร เนื่องจากเป็นขั้นตอนที่สำคัญและจำเป็นอย่างหนึ่งที่จะช่วยให้งานเขียนคอมพายเลอร์เป็นไปอย่างมีแบบแผน และสามารถใช้ประโยชน์ในกรณีที่มีความจำเป็นต้องเปลี่ยนแปลง แก้ไขหรือพัฒนาในโอกาสต่อไป รูปแบบของเอกสารที่จัดทำขึ้นอาจอยู่ในรูปแบบใดก็ได้ เช่น เป็นตารางแสดงความสัมพันธ์ของเหตุการณ์ต่างๆ หรือเป็นภาคผนวกที่ใช้บรรยายสิ่งที่ควรรู้ แต่ยังไม่ได้กล่าวไว้ในเอกสารที่เป็นเนื้อหา หรือแม้แต่เป็นลักษณะของดัชนีเพื่อการค้นหาข้อความที่เป็นจุดสำคัญของเนื้อหา วัตถุประสงค์หลักของการจัดทำเอกสารนั้นก็เพื่อบันทึกสิ่งต่างๆ เพื่อเป็นการช่วยงานเขียนคอมพายเลอร์ ทั้งในระหว่างการดำเนินงานและการบำรุงรักษาหรือพัฒนาให้มีประสิทธิภาพสูงขึ้น

ตัวอย่างของการนำเทคนิคแบบแบ่งงานออกเป็นส่วนย่อยๆ มาใช้ก็ได้แก่การแยกเอาบางส่วนของคอมพายเลอร์มาอธิบายเช่นส่วนที่เป็น parser เป็นต้น ตามที่เราได้พูดถึงกันมาแล้วว่า การจัดทำเอกสารนั้นก็เพื่อให้เกิดความเข้าใจในงานทั้งหมดที่ต้องทำและเพื่อเตือนความจำ ดังนั้นส่วนต่างๆ ของเอกสารอาจประกอบด้วยเรื่องราวของประวัติความเป็นมาของการเขียนคอมพายเลอร์ที่ได้ออกแบบไว้ การให้คำจำกัดความของภาษา (source language) ชนิดของระบบคอมพิวเตอร์ที่ใช้ในการเขียนและที่จะนำคอมพายเลอร์ไปใช้ การสำรวจความต้องการของตลาดและอาจรวมถึงการประเมินผลตอบแทนจากการลงทุนของการสร้างคอมพายเลอร์ (ในกรณีที่ทำเพื่อการค้า) เป็นต้น จะเห็นได้ว่างานด้านการจัดรูปแบบของเอกสารนั้นนอกจากจะช่วยในการจัดลำดับขั้นตอนและจัดกลุ่มเพื่อให้เป็นไปอย่างเหมาะสมแล้ว ก็ยังเป็นเครื่องมือที่ช่วยให้เกิดความเข้าใจต่องานเขียนคอมพายเลอร์ได้ดีขึ้นอีกด้วย

อย่างที่เราเคยได้พูดถึงกันแล้วว่าโปรแกรมคอมพายเลอร์นั้นมักเป็นโปรแกรมที่มีขนาดใหญ่ ซึ่งอาจต้องใช้บุคคลหลายคนช่วยกันเขียนเพื่อให้งานเสร็จเร็วและมีประสิทธิภาพมากขึ้น ดังนั้นงานในด้านการจัดการเพื่อแบ่งสรรงานให้กับลูกทีม จึงมีความจำเป็นต่องานเขียนคอมพายเลอร์มากพอควร และการนำเทคนิคนี้มาใช้จะต้องตั้งอยู่บนหลักเกณฑ์ของการที่จะพยายามลดความยุ่งยากในการทำงานให้มากที่สุด ในขณะที่จะต้องให้คุณภาพของงานดีที่สุดด้วยเช่นกัน

การแบ่งงานเขียนออกเป็นส่วนๆ เพื่อมอบหมายให้กับผู้เขียนโปรแกรมแต่ละคน (modular programmer assignment) นั้น จะต้องเป็นส่วนที่มีขั้นตอนการทำงานซึ่งสามารถแยกออกไปได้อย่างอิสระไม่ว่าจะเป็นด้านการเขียน การแก้ไข ปรับปรุง ทดสอบหรือการพัฒนาในส่วนนั้นๆ ซึ่งส่วนต่างๆ ที่แบ่งไว้ อาจเป็นหน้าที่ในส่วนใดส่วนหนึ่งของคอมพายเลอร์ แต่ผลลัพธ์ที่ได้จะต้องสามารถนำมาใช้ด้วยกันได้อย่างต่อเนื่อง และจากเหตุผลที่ว่าการพัฒนาโครงสร้างหรือส่วนต่างๆ ของคอมพายเลอร์นั้นจะเกิดขึ้นระหว่างการดำเนินการ ซึ่งจะทำให้ลำดับขั้นตอนในการทำงานอาจเปลี่ยนแปลงไป ทั้งนี้ขึ้นอยู่กับการพิจารณาว่าควรจะทำอะไรก่อนหลัง ดังนั้นวิธีการแบ่งโครงสร้างการทำงานออกเป็นส่วนๆ จะช่วยให้ผู้ออกแบบสามารถเห็นภาพรวมของงานและตัดสินใจได้ว่าควรจะดำเนินการในส่วนใดก่อน ซึ่งหมายความว่าอาจจะให้มีการทำงานเป็นลำดับขั้นจากบนลงล่าง (top-down) สำหรับบางส่วนของงาน ในขณะที่อาจต้องใช้วิธีการแบบจากข้างล่างขึ้นข้างบน (bottom-up) ในบางส่วนของงานด้วยก็ได้

ในการเขียนคอมพายเลอร์ในส่วนของโปรแกรมต้นฉบับด้วยการแบ่งโปร



แกรมออกเป็นส่วนๆ (modular source text) ตามลักษณะการทำงานหรือหน้าที่ของคอมพายเลอร์นั้น โดยปกติแล้วจะเป็นการจัดแบ่งตามกลุ่มของโปรแกรมย่อยที่มีลักษณะการทำงานเกี่ยวข้องกัน หรือมีการใช้ข้อมูลร่วมกันเข้าไว้ในโมดูลเดียวกัน อย่างไรก็ตามการแบ่งส่วนของโปรแกรมควรจะคำนึงถึงขนาดและการติดต่อสื่อสารกันภายในโมดูลด้วย ซึ่งขนาดของโมดูลควรจะเล็ก และเรียกใช้โปรแกรมย่อยต่างๆ น้อยที่สุด ทั้งนี้เนื่องจากโมดูลที่มีขนาดใหญ่ขึ้น ก็ย่อมทำให้การตรวจสอบ แก้ไข หรือการดำเนินการต่างๆ นั้น มีความยุ่งยากมากขึ้น และการที่มีการติดต่อสื่อสารหรือเรียกใช้โปรแกรมย่อยมากครั้งขึ้น ก็จะทำให้ต้องเสียเวลาในการปฏิบัติงานของคอมพายเลอร์เพิ่มขึ้นด้วย ดังนั้นการกำหนดโครงสร้างในลักษณะนี้ จะต้องคำนึงถึงความเร็วในการทำงานของคอมพายเลอร์ควบคู่กันไปด้วย

โปรแกรมคอมพายเลอร์เป็นโปรแกรมที่มีขนาดใหญ่ ซึ่งบางครั้งการที่จะประมวลผลทั้งโปรแกรมอาจทำไม่ได้ โดยเฉพาะอย่างยิ่งปัญหาเกี่ยวกับความจุของหน่วยความจำหลัก ดังนั้นโปรแกรมคอมพายเลอร์ที่เป็นโปรแกรมตามเป้าหมายก็มักจะถูกแบ่งออกเป็นส่วนย่อยๆ (modular target text) เช่นเดียวกัน โดยที่ส่วนต่างๆ ที่แบ่งออกไปนั้น จะต้องสามารถทำงานได้อย่างอิสระหรือมีการเรียกใช้โมดูลอื่นๆ น้อย และในการเขียนคอมพายเลอร์จริงๆ นั้นก็มักจะเขียนในลักษณะ multipass compiler ซึ่งหมายความว่าจะมีการเรียกคอมพายเลอร์เข้าไปเก็บไว้ในหน่วยความจำหลักเฉพาะส่วนที่ต้องการใช้ในขณะใดขณะหนึ่งเท่านั้น ทำให้ช่วยประหยัดเนื้อที่ของหน่วยความจำได้มาก

เมื่อจะให้โมดูลต่างๆ คุยกัน

การที่เราได้แยกโปรแกรมใหม่ออกเป็นส่วนย่อยๆ นั้น ปัญหาที่จะตามมาทันทีก็คือ การติดต่อกันระหว่างโปรแกรมเหล่านั้น ลองมาดูกันต่อไปว่าจะทำยังไงกันดี ในกระบวนการทำงานของโมดูลต่างๆ นั้น จะมีลักษณะที่ต้องมีความเกี่ยวข้องสัมพันธ์กัน ซึ่งหมายความว่าจะต้องมีขั้นตอนในการติดต่อกันระหว่างโมดูลที่ต้องใช้งานร่วมกัน รวมถึงการติดต่อสื่อสารกันภายในโมดูลเอง โดยที่การติดต่อกันนั้นจะทำโดยการผ่านข้อมูลระหว่างกันและมีรูปแบบที่แน่นอน ซึ่งจะต้องกำหนดโครงสร้างของข้อมูลที่จะใช้ติดต่อกันไว้ก่อน และการกำหนดนั้นสามารถทำได้ทั้งในลักษณะของไดนามิกที่อาจทำการเปลี่ยนขนาดของข้อมูลได้ในขณะประมวลผล หรือแบบสแตติกที่กำหนดรูปแบบและขนาดไว้ตายตัว เช่น ในลักษณะของตารางหรือสตริงที่มีความยาวคงที่

การกำหนดขั้นตอนในการติดต่อกันระหว่างโมดูลนั้น ผู้เขียนโปรแกรมจะต้องพิจารณาด้วยความรอบคอบเพื่อเป็นการป้องกันความผิดพลาดที่อาจจะเกิดขึ้นจากการส่งผ่านข้อมูลระหว่างกัน ที่สามารถเกิดขึ้นได้ง่ายกว่าการที่จะติดต่อกันภายในโปรแกรมเดียวกัน ซึ่งความระมัดระวังและการกำหนดรูปแบบการทำงานอย่างเหมาะสมนั้น นอกจากจะทำให้การทำงานของคอมพายเลอร์เป็นไปอย่างถูกต้องแล้ว ยังเป็นการประหยัดเวลาในการประมวลผลอีกด้วย

สิ่งที่จะละเลยไม่ได้ในการเขียนโปรแกรมคอมพิวเตอร์ (ไม่ว่าจะมีลักษณะเป็นโปรแกรมเดียวโดดๆ หรือเป็น โมดูลก็ตาม) ก็คือการทดสอบการทำงานของโปรแกรม โดยเฉพาะอย่างยิ่งการเขียนโปรแกรมในลักษณะของโมดูลนั้น การทดสอบจะต้องทำด้วยความระมัดระวัง เนื่องจากข้อผิดพลาดที่เกิดจากการติดต่อหรือส่งผ่านข้อมูลระหว่างกันสามารถเกิดขึ้นได้ง่ายกว่า การทดสอบด้วยความรอบคอบนอกจากจะเป็นการตรวจสอบเพื่อแก้ไขข้อผิดพลาดของโปรแกรมแล้ว ยังเป็นวิธีการที่สามารถหาข้อผิดพลาดได้เร็วที่สุดด้วย (แทนที่จะปล่อยให้ใช้โปรแกรมไปเรื่อยๆ และพบข้อผิดพลาดในภายหลัง) เพราะฉะนั้นการทดสอบจึงจำเป็นต้องใช้ข้อมูลในลักษณะต่างๆ (ทั้งที่ถูกต้องและไม่ถูกต้อง) และเป็นจำนวนที่มากพอ รวมถึงข้อมูลที่จะใช้จริงในการทำงานด้วย หลังจากการทดสอบด้วยข้อมูลต่างๆ เรียบร้อยแล้วก็ควรจะทดสอบด้วยข้อมูลจริงในช่วงเวลาหนึ่งเพื่อให้เกิดความมั่นใจมากขึ้น

ออกแบบโครงสร้างการทำงาน

อีกสิ่งหนึ่งที่จะละเลยในการพิจารณาไม่ได้ก็คือ การกำหนดโครงสร้างในการทำงานของคอมพายเลอร์ เพราะการกำหนดโครงสร้างที่เหมาะสม ย่อมหมายถึงการเพิ่มสมรรถนะการทำงานให้กับคอมพายเลอร์ด้วย ซึ่งรูปแบบการทำงานของ

COMPILE IN THE FAST LANE

โปรแกรมสามารถแบ่งออกได้เป็นสองลักษณะคือ การกำหนดให้มีการกระจายตามแนวดิ่งและตามแนวระดับ การทำงานของโปรแกรมในลักษณะที่มีขั้นตอนการทำงานตามแนวดิ่ง (Vertical Fragmentation) นั้น เป็นการทำงานในรูปแบบของโปรแกรมที่สามารถแบ่งออกเป็นโปรแกรมย่อยๆ ได้ (multipass structure) ลักษณะการทำงานจะมีรูปแบบพื้นฐานตามรูปที่ 5 ซึ่งโปรแกรมแบ่งออกเป็น 7 โมดูล เริ่มตั้งแต่การให้ input เพื่อทำการตรวจสอบความหมายและความถูกต้อง ทำการประมวลผล และให้ผลลัพธ์ออกมาในขั้นตอนสุดท้าย

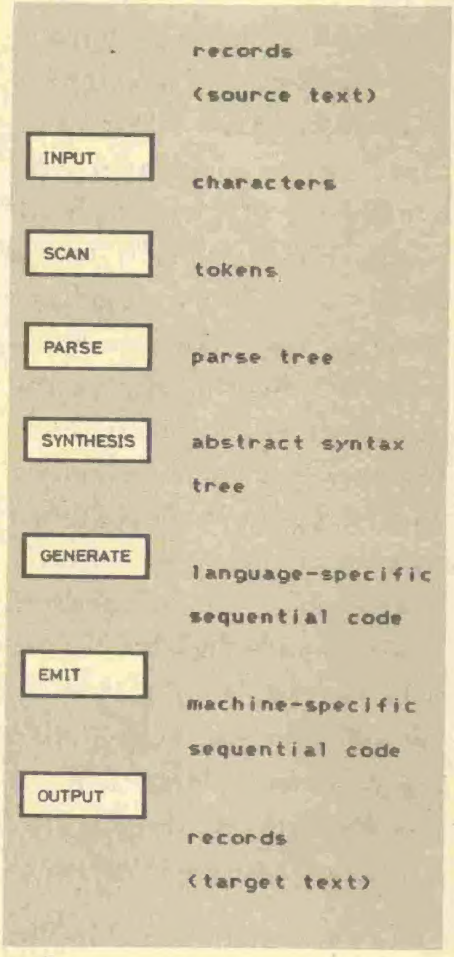
เพื่อให้เห็นได้ชัดเจนขึ้นเราลองมาสมมุติกันว่า มีคำสั่งอยู่ 1 ประโยค และให้มีการเขียนโปรแกรมคำสั่งด้วยบัตรคอมพิวเตอร์ดังนี้

IF X V = 1 THEN X = 53 + X

การทำงานของคอมพายเลอร์ในลักษณะของแนวดิ่งตามขั้นตอนในรูปที่ 5 นั้น ก็จะเริ่มจากขั้นตอนแรก **(INPUT)** โดยรับชุดคำสั่งที่เป็น characters ต่างๆ เข้ามาแล้วทำการตรวจสอบและแยกแยะว่าประกอบด้วยอะไรบ้าง (เช่น เป็นคำสั่ง ตัวแปรค่าคงที่ เครื่องหมายคำนวณ ฯลฯ ซึ่งในที่นี้เรียกว่า token) **ในขั้นตอนที่สอง (SCAN)** และตามตัวอย่างนี้ก็จะแยกออกได้เป็น IF, X, V, =, 1, THEN, X, =, 53, +, และ X โดยจะทำการเก็บ token เหล่านี้เอาไว้ในตาราง ต่อจากนั้นก็จะทำการกระจายเพื่อตรวจสอบความถูกต้องของกฎเกณฑ์ภาษา ซึ่งเป็นขั้นตอนที่สาม **(PARSE)** แล้วจะทำการแทนที่เครื่องหมายต่างๆ ด้วยคำสั่งภาษา Assembly ซึ่งเป็นขั้นตอนที่สี่ **(SYNTHESIS)** จากนั้นก็จะทำการสร้างรหัสคำสั่งในขั้นตอนที่ห้า **(GENERATE)** ขั้นตอนต่อไปซึ่งเป็นขั้นตอนที่หก **(EMIT)** ก็จะทำการแปลรหัสที่ได้ให้เป็นชุดคำสั่งที่เครื่องสามารถนำไปประมวลผลได้ และจะทำการแปลให้เป็นรหัสภาษาเครื่องในขั้นตอนสุดท้าย โดยขั้นตอนต่างๆ เหล่านี้ได้แสดงไว้ในรูปที่ 6

ในการกระจายขั้นตอนการทำงานลักษณะนี้ เทคนิคแบบ tree นับเป็นวิธีการที่ถูกนำมาใช้เป็นหลักสำคัญ ขั้นตอนต่างๆ ที่เกิดขึ้นจากการทำงานของคอมพายเลอร์จะทำให้โครงสร้างของ tree เปลี่ยนแปลงไป ซึ่งอาจเป็นการตัดทอนเพิ่มเติม คงไว้ หรือเปลี่ยนรูปแบบออกไปตามความเหมาะสม ดังนั้น การเขียนคอมพายเลอร์ด้วยวิธีการแบ่งขั้นตอนการทำงานออกเป็นโมดูลที่ทำงานต่อเนื่องกันไปนั้น ก็จำเป็นจะต้องรู้ถึงวิธีการนำเทคนิคแบบ parse - structure tree มาใช้ด้วย ซึ่งประโยชน์ที่จะได้รับจากการใช้เทคนิคนี้ก็คือ ทำให้งานแปลในส่วนที่เหลือง่ายขึ้น และทำให้การปรับปรุงผลลัพธ์ที่ได้ให้อยู่ในรูปแบบที่เหมาะสมกว่าเดิมทำได้สะดวก

การทำงานอีกรูปแบบหนึ่งที่แตกต่างออกไปจากการทำงานในแต่ละขั้นตอน (module) ให้เสร็จสิ้นก่อนแล้วจึงทำขั้น



รูปที่ 5 แสดงขั้นตอนการทำงานตามแนวดิ่ง

ตอนอื่นต่อไป ตามวิธีการแบบ Vertical Fragmentation นั้น ได้แก่การทำงานหลายๆ โมดูลพร้อมกันไป ซึ่งวิธีการนี้เรียกว่า Horizontal Fragmentation และวิธีการนี้นับว่ามีประโยชน์มากในการเขียนโปรแกรมคอมพายเลอร์ การใช้เทคนิคแบบนี้ก็ได้แก่การเรียกใช้โปรแกรมย่อยต่าง ๆ เช่น subroutine หรือ subprogram ตลอดจนโมดูลอื่นๆ การเรียกใช้โปรแกรมย่อยจากโปรแกรมหลักจะสามารถทำได้ในขณะที่กำลังประมวลผลโปรแกรมหลักอยู่ โดยการเลือกที่จะทำงานอย่างใดอย่างหนึ่ง ซึ่งอาจมีหลายๆ กรณี ให้เลือกทำได้เป็นต้น

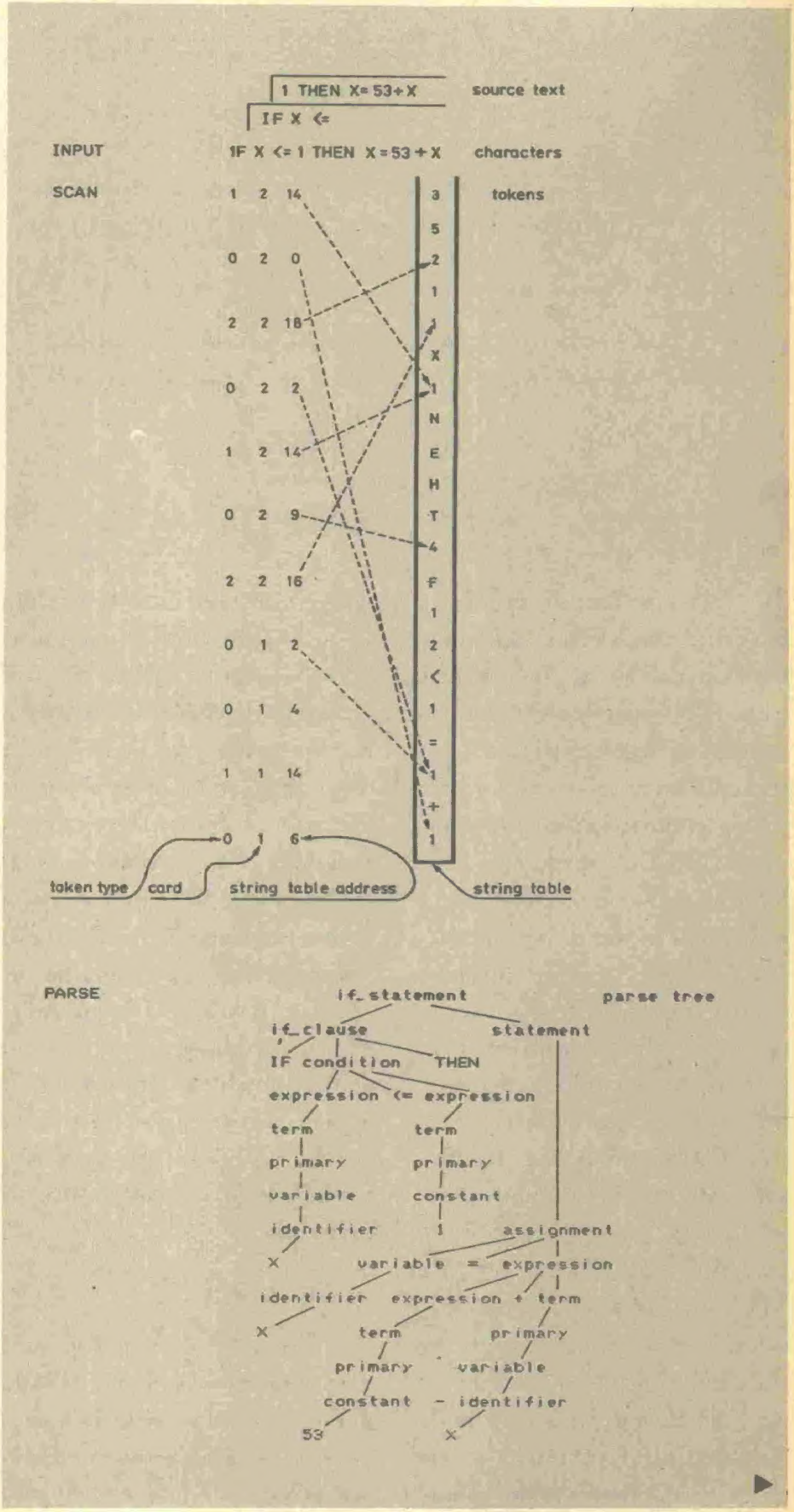
วิธีการแบบนี้จะช่วยให้สามารถกำหนดทางเลือกไว้ได้หลายๆ ทางตามแต่เงื่อนไขที่แตกต่างกันออกไป ซึ่งจะทำให้การเขียนโปรแกรมคอมพายเลอร์ (รวมทั้งโปรแกรมอื่นๆ) มีโครงสร้างที่ดี และมีการทำงานอย่างเป็นขั้นตอน สามารถตรวจสอบแก้ไขข้อผิดพลาดรวมถึงการพัฒนางานในโอกาสต่อๆ ไป ทำได้สะดวกยิ่งขึ้น

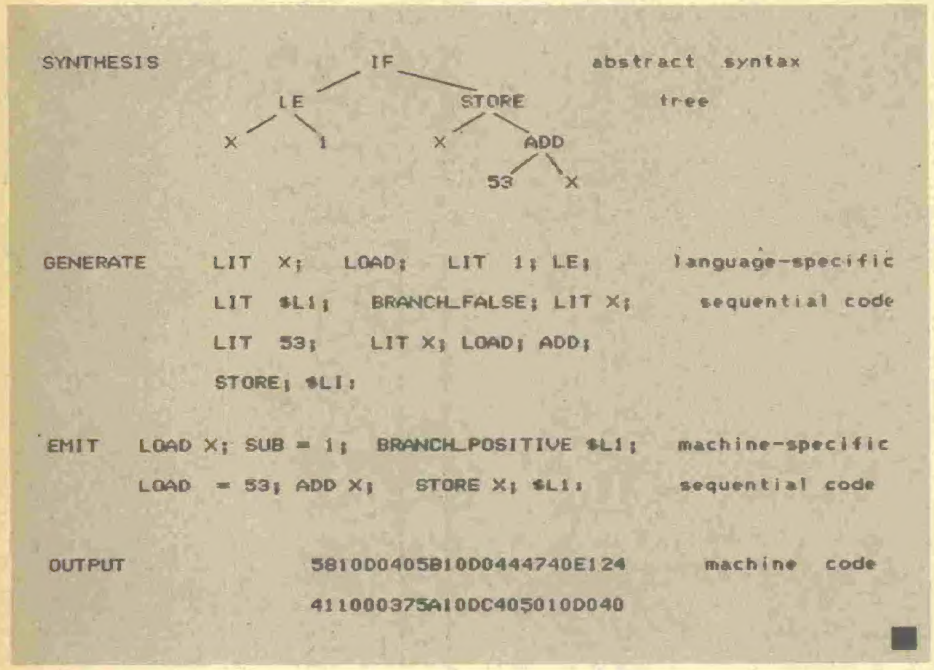
ดังนั้นในการเขียนโปรแกรมหรือการจัดโครงสร้างของคอมพายเลอร์จึงมีความจำเป็นจะต้องใช้วิธีการแบ่งขั้นตอนการทำงานทั้งสองลักษณะคือทั้งตามแนวดิ่ง และแนวราบ โดยจะต้องพิจารณาถึงความเหมาะสมของขั้นตอนการทำงานของโปรแกรม ว่าเมื่อใดควรจะใช้วิธีการอย่างใด ซึ่งถ้าผู้เขียนสามารถวางโครงสร้างต่างๆ ได้อย่างเหมาะสมแล้วก็จะทำให้โปรแกรมคอมพายเลอร์ที่เขียนขึ้นนั้นมีขั้นตอนการทำงานที่เป็นไปอย่างสอดคล้องกลมกลืนกัน การทำงานของคอมพายเลอร์ก็จะเป็นไปอย่างมีประสิทธิภาพมากขึ้น

การประเมินผล

สิ่งสุดท้ายที่จะขาดไปไม่ได้ก็คือการประเมินผลได้ผลเสียในการที่จะทำการเขียนคอมพายเลอร์ที่คิดไว้นั้นขึ้นมา

รูปที่ 6 แสดงขั้นตอนการทำงานแบบกระจายตามแนวดิ่ง





ว่า มีความเหมาะสมมากน้อยเพียงใด เพราะการเขียนคอมพายเลอร์ก็เหมือนกับการดำเนินงานทั่วๆ ไป คือ ต้องมีการลงทุน (ซึ่งอย่างน้อยที่สุดก็เป็นค่าแรงงานของคนเขียนโปรแกรม ค่ากระดาษ ค่าปากกา ใช่ไหมครับ) โดยเฉพาะอย่างยิ่งการเขียนคอมพายเลอร์ดีๆ ขึ้นมาสักตัวหนึ่งนั้นไม่ใช่เป็นการลงทุนเล็กๆ น้อยๆ แน่ เพราะฉะนั้น ย่อมจะหลีกเลี่ยงการที่จะต้องคิดถึง ความคุ้ม ในการตกลงใจไปไม่ได้ ซึ่งคำว่า"ความคุ้ม"ในที่นี้ย่อมจะต้องรวมถึงทั้งในแง่เศรษฐกิจจริงๆ (คิดถึงเรื่องกำไร - ขาดทุน ที่จะเกิดขึ้นจากงานชิ้นนั้นๆ) และในส่วนที่เป็นความพอใจของผู้ตัดสินใจเอง (ซึ่งในกรณีนี้ไม่สามารถจะกำหนดเป็นมาตรฐานได้ เนื่องจากสิ่งที่บางคนเห็นว่าเป็นผลประโยชน์หรือผลกำไร อาจจะเป็นต้นทุนของคนอื่นๆ ก็ได้) อย่างไรก็ตามความคุ้มหรือไม่ก็จะเห็นได้ชัดเจนขึ้น ถ้าเราจะทำการเปรียบเทียบกับการนำทรัพยากรเท่าที่มีอยู่นั้นไปใช้ดำเนินการในงานด้านอื่นๆ

สำหรับปัญหาที่จะนำมาพิจารณาในการตัดสินใจนั้นก็ควรจะประกอบด้วยประโยชน์ที่เกิดขึ้นจากการได้สร้างคอมพายเลอร์ขึ้นมาว่าจะมีอะไรบ้าง (เช่น ทำให้เจ้าหน้าที่มีงานทำ และเป็นการเสริมสร้างประสบการณ์ให้แก่เจ้าหน้าที่เป็นต้น) คุณประโยชน์ที่ได้รับโดยตรงจากคอมพายเลอร์นั้น มีมากน้อยแค่ไหนเมื่อเทียบกับสิ่งที่มีอยู่เดิม (เช่นสามารถสร้างรหัสภาษาเครื่องที่ทำงานได้รวดเร็ว ถูกต้องแม่นยำ มีความต้องการใช้หน่วยความจำเพียงเล็กน้อย และสามารถทำงานได้ตรงตามวัตถุประสงค์ที่วางเอาไว้เป็นต้น) ตลอดจนผลิตผลที่จะได้จากคอมพายเลอร์ (ซึ่งอาจมีการนำไปใช้ประโยชน์ได้อย่างกว้างขวาง ทำให้เกิดคุณประโยชน์ต่องานด้านการเขียนโปรแกรมคอมพิวเตอร์) สิ่งต่างๆ เหล่านี้ล้วนแล้วแต่มีผลกระทบและเป็นเหตุผลที่จำเป็นในการช่วยให้การตัดสินใจเป็นไปด้วยความเหมาะสมยิ่งขึ้นทั้งสิ้น

คุยกันก่อนจบ

ในคราวนี้เราก็ได้คุยกันถึงเรื่องราวต่างๆ หลายอย่างที่เกี่ยวข้องกับสิ่งที่จำเป็นต้องรู้หรือควรที่จะต้องคำนึงถึง เมื่อเราต้องการที่จะเขียนคอมพายเลอร์ไม่ว่าจะเป็นเรื่องของเทคนิคต่างๆ ที่จะต้องนำมาใช้ เช่น การเรียกใช้ตัวเอง หรือการแบ่งงานออกเป็นส่วนๆ ตลอดจนสิ่งที่ควรรู้ต่างๆ เช่น การกำหนดรูปแบบของภาษา การกำหนดโครงสร้างการทำงานของคอมพายเลอร์ หรือแม้แต่การประเมินผลเพื่อดูว่าจะคุ้มค่าหรือไม่ สำหรับการตัดสินใจทำงานชิ้นนั้น ซึ่งเรื่องราวต่างๆ ที่เราได้พูดถึงนี้เป็นสิ่งที่ไม่ควรจะมองข้ามไป ถ้าหากต้องการผลงานที่ดีและคุ้มค่าต่อการลงทุน

งานเขียนคอมพายเลอร์เป็นงานใหญ่ที่ต้องการการวางแผนที่รอบคอบรัดกุมและเหมาะสม โดยปกติแล้วปัญหาในการทำงานนั้นมักจะแปรผันตามขนาดและปริมาณงาน ดังนั้นการวางแผนที่ดีย่อมหมายถึงความราบรื่นในการทำงานและประสิทธิภาพของงานที่จะได้รับเพราะการวางแผนงานอย่างมีระบบ มีขั้นตอนและโครงสร้างที่ชัดเจนจะเป็นการช่วยขจัดอุปสรรคหรือแก้ปัญหาต่างๆ ที่อาจเกิดขึ้นในระหว่างการดำเนินงานได้มาก และก็คงจะไม่เป็นการเกินเลย ถ้าจะพูดว่าการวางแผนงานที่ดี ย่อมหมายถึง การทำงานเสร็จไปแล้วครึ่งหนึ่ง อย่างน้อยที่สุดก็ช่วยให้เกิดความมั่นใจได้ว่า งานที่ทำนั้นจะไม่ต้องเลิกล้มกลางคัน เพราะฉะนั้น ถ้าคุณเป็นคนหนึ่งที่สนใจจะเขียนคอมพายเลอร์ละก็ อย่าลืมเรื่องที่เราคุยกันในครั้งนี้นะครับ สวัสดี.

เอกสารอ้างอิง

Alfred V. AHO and Jeffrey D. Ullman **"Principles of Compiler Design,"** Addison-Wesley Pulishing Company, 1978.

F.L Bauer and J. Eickel, (Editors), **"Compiler Construction,"** Springer-Nerlag Berlin, 1976.

Philip M. Lewis II and Others, **"Compiler Design Theory,"** Addison-wesley Publishing Company, 1976.